

Incremental Migration from Enscribe to SQL: The Database Gateway Solution

Dr. Richard Carr, Ph.D.
Carr Scott Software Incorporated
Cupertino, California

Published in Volume 18, Number 4, of the Tandem Connection, the official magazine of the International Tandem Users' Group. Copyright 1997, International Tandem Users' Group.

The Enscribe file system continues to be one of the most reliable and efficient database systems for enterprise computing, but the typical Enscribe user has a strong desire to move to a more modern database technology. The primary drawbacks of Enscribe are its lack of openness, poor data accessibility, and limits on the size and manageability of the database.

Openness and Standards. Enscribe is clearly limited to the Tandem NSK platform; there are no easy methods for porting applications between Enscribe and other platforms. There are no existing tools or systems that permit the straightforward combination of data between Enscribe and other database systems.

Database Accessibility. For many years, the only application-level tools for Enscribe were Enform and Enable. The Enform query processor does not support database updates. Both are limited to 6530 terminal interfaces. Although there are a few GUI 3rd-party Enscribe database tools available, they pale in comparison with workstation-based tools that operate on SQL/ODBC databases.

Manageability and Limits. Typical Enscribe database maintenance operations, such as adding a field to a record is a major undertaking; re-partitioning a file requires making the data unavailable for a lengthy operation. The maximum of 16 partitions limits both maximum file size and scalability. The use of Enscribe for massive data storage would be impractical.

“A legacy information system is one that significantly resists modification and change.” (Brodie, M., and Stonebraker, M., 1995) Most Enscribe-based applications can be described as legacy information systems.

NonStop SQL/MP

For most owners of Enscribe-based systems, the obvious migration target is Tandem's NonStop SQL/MP, which has improved on Enscribe's fault-tolerance, scalability, efficiency and reliability. Migrating to NonStop SQL/MP requires much less re-engineering and organizational chaos than any other alternative.

Data Access. Data in a SQL database is universally accessible through ODBC and non-programmer tools such as PC/Mac spreadsheets and word-processors. SQL is the universal solution to the data access problem.

Data Management. Databases must change to reflect changing reality. SQL supports adding columns to a table without changing the existing data or affecting existing programs. Programs that access or update a table operate on specific columns, not records; other columns are ignored for access, and are preserved through updates.

A recent *Tandem Connection* article (Alleman, 1997) describes the considerable effort required to add a few fields to an Enscribe file with minimum outage (about an hour); the equivalent SQL operation is accomplished in seconds.

Dr. Richard Carr has been a designer and developer of Tandem-based system software since 1981. As head architect of NSK / Guardian development in the early 1980's, Dr. Carr was author of the memory manager, expedited messages, parallel reloading, DSAP and DCOM. He received Tandem's first software patent for his work on the global update protocol. In later years at Tandem, at the High Performance Research Center and as Technical Director at Tandem Labs, he mixed advanced development with large customer consulting and competitive benchmarking. His last project was RDF/MP, a.k.a. Turbo RDF. In 1995, Dr. Carr founded Carr Scott Software to build the database gateway described in the accompanying article.

SQL also supports on-line move and split partition, which are causes of lengthy Enscribe file outages.

Data Manipulation. Programming complex database operations is considerably easier in SQL than in legacy databases. SQL is deservedly famous for its ad-hoc query and update capability.

Dealing with massive amounts of data (decision support, data mining, etc.) is a job well-suited for SQL. NonStop SQL/MP performs query optimization and then employs parallelism, hash joins, large data transfers, multidimensional data access, etc., to reduce the query times by orders of magnitudes, without complex programming.

Data Validation. SQL guarantees that each value in the database is consistent with the defined data type and any constraints. Each column in a table may require a data value, have a default value, or default to the NULL value.

Standards-based Programming. It is difficult to obtain programming skills for any legacy system. Those who have such skills are concerned about career development; attracting and training competent Enscribe programmers is a challenge. On the other hand, talented SQL programmers are relatively easy to hire and retain.

Traditional Database Migration

The traditional migration from a legacy system to SQL requires examining each program that operates on the database, understanding the logic that performs database operations, and replacing that logic with SQL data access statements. Once the programs are rewritten, and tested, the legacy files are converted to SQL tables and, all at once, the new SQL-based applications are placed into production and the old legacy applications are retired.

The process may not be technically difficult and the projected cost of programming may be affordable, but many users have studied and abandoned plans to migrate in this fashion, due to the unacceptable risk and impact it would have on the enterprise. The following sections outline some of the problems with a traditional migration.

High Conversion Cost. There are no known methods of reliably converting source programs from Enscribe to SQL using mechanical translators. Studies of other legacy program migration (e.g., Year 2000) estimate the cost of manual conversion at dollars per line of original source code; SQL conversion is more complex than Year 2000 conversion.

It is often the case that a large percentage of programs are functionally stable, and converting them to SQL would not enhance the users ability to do business. The traditional approach, however, demands converting every program that *might* be needed after the database conversion.

High Conversion Risk. The legacy system was built incrementally; new programs or upgrades were placed in service individually; bugs found after introduction could be quickly corrected without major impact. Placing hundreds or thousands of rewritten programs in service all at the same time, even if they are well tested, is an incredibly risky step for any business. Even if the bug rate is quite low, the number of problems may quickly overwhelm the ability of the organization to deal with them.

It may be possible to decompose the application into separate collections of files and the programs that operate on them, and convert one collection at a time, but few applications were designed to permit such a decomposition.

Large Project Risks. A large and risky migration project will not be approved if it proposes a complete rewrite of the existing system with no immediate return on investment. The project must promise new function or significant cost saving. Thus, the project must incorporate broad logic changes to the application, further increasing the cost and risk.

Management of a large project over a long time is hard. Business conditions will change. Since there may be no absolute requirement to migrate the applications, the organization must constantly rejustify why money is being spent for a long-term goal.

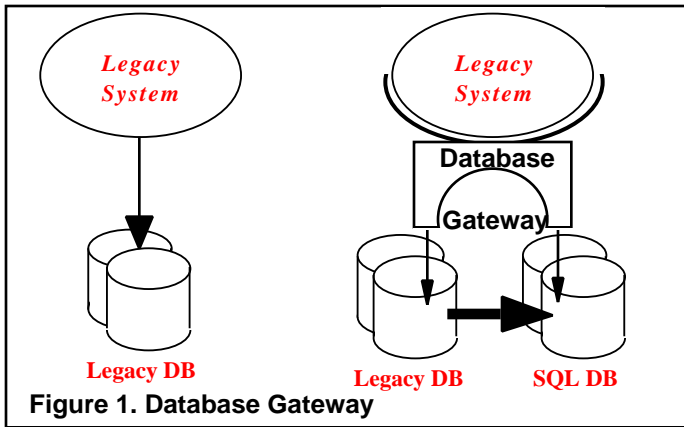
Also, large migration projects have a host of additional problems: they require cooperation of people with an intellectual investment in the legacy system; they open up opportunities to reexamine the entire design of the legacy system; their true cost and schedule are hard to estimate accurately. These perils of the large migration project are described in Brodie's and Stonebraker's book, *Migrating Legacy Systems*.

Impact on Critical Development. Any major re-engineering project will require careful and costly coordination with ongoing development required by the business. The legacy application must continue to serve the needs of the business, requiring the senior architects to split their attention between ongoing development and the re-engineering process.

Lost Opportunities. If the primary goal of the migration is to provide better access to the data, or to program new application function in SQL, realizing the goal is delayed until the re-engineering project is completed.

Database Gateways

A database gateway is an agent that insulates the existing application programs from changes that are made to the database. A simple illustration of a database gateway appears in Figure 1.



The gateway supports conversion of the physical data from the legacy database to the desired target SQL database. When legacy programs are executed, the gateway emulates the legacy API using data from the SQL database. New programming and standards-based tools can access the SQL database directly.

The gateway should support the ability to add new function and to modernize the database structure (such as adding or altering field definitions and normalizing files), but without affecting the existing legacy programs.

Later sections will describe the construction, operation, and use of an Enscribe-to-SQL gateway, but first let us consider its advantages over traditional migration.

Reduced cost and risk. Although the gateway can eliminate most reprogramming, this may be a minor consideration. Even if one reprograms all legacy programs, the gateway can reduce overall project costs.

Once the database is converted, any reprogramming effort can be concentrated in the most fruitful area first. New development or performance-critical programs can be addressed first, followed by the other programs in priority order. The user can continually reassess the value of reprogramming the entire application vs. leaving seldom-executed programs as-is.

Improved productivity. One can form a SWAT team to focus solely on the migration task; such a team can concentrate on a few programs at a time, reducing the need to coordinate with ongoing development. They will develop skills and methods from the first few programs, which can accelerate the entire process.

As parts of the application are reprogrammed and tested, they can be installed on the production system gradually, reducing the immediate risk to a more acceptable level.

Accelerated return on investment. A large portion of the value of a SQL database is its increased accessibility with modern tools and by non-programmers. The gateway converts the database immediately; its cost is often justified by the accelerated availability of the data.

Escort SQL

Escort SQL is a database gateway that migrates Enscribe application programs and Enscribe files to NonStop SQL/MP. The gateway has three primary facilities: table mapping, data migration, and Enscribe emulation.

Table mapping uses the available information about Enscribe files, typically DDL records and the existing file attributes; the user may specify additional instructions to restructure the target tables, including altering data types, specifying data scrubbing, Y2000 date widening, or proprietary data conversion, and normalizing the file. The table and indexes, if any, are created, and the mapping information is stored for later use.

Data migration accesses the table mapping information and converts the Enscribe data records to load the target SQL tables.

Enscribe emulation intercepts all Enscribe operations on SQL tables, accesses the table mapping, and performs equivalent SQL operations.

Gateway Usage Overview

The following figures illustrate the typical steps to use the gateway.

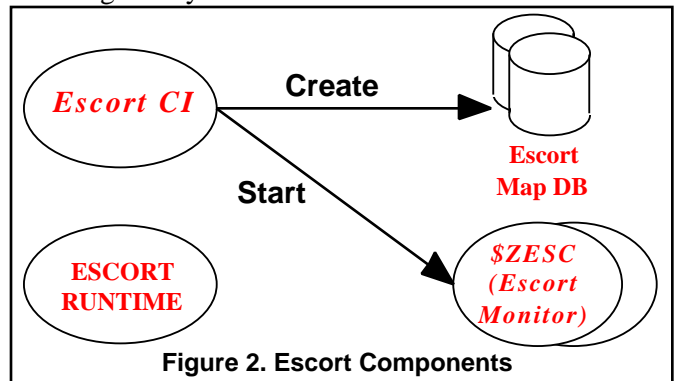


Figure 2. shows the product components after the initial installation, which is a simple restore of the product files. The *Escort CI* is the command interface to manage the gateway; it is used to create the *map database* that holds the table mapping information. The *runtime* is a user library that performs Enscribe emulation. The *monitor* is a NonStop process pair that performs coordination and caching; it is configured and started by the CI.

The gateway components are non-privileged, use only standard Tandem interfaces, do not require a SYSGEN or SUPER.SUPER access, and conform to both standard or SafeGuard security.

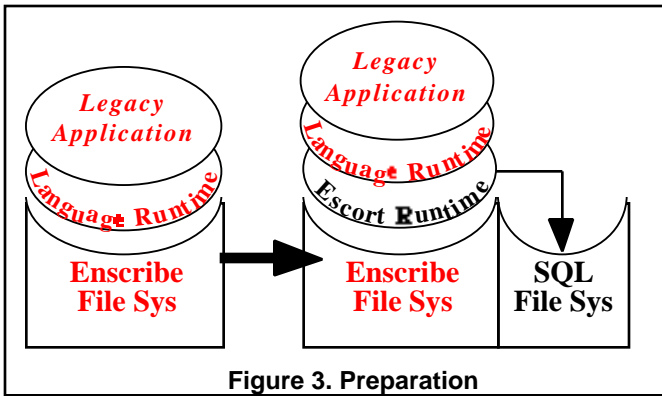


Figure 3. Preparation

Application preparation is shown in Figure 3. For each application program, an automated process inserts the gateway runtime between the program and Enscribe. This one-time operation takes a few seconds for each program.

Prepared programs do not require any special handling; they do not need to be SQLCOMPIled, even if they are duplicated, moved, renamed, or copied to another system.

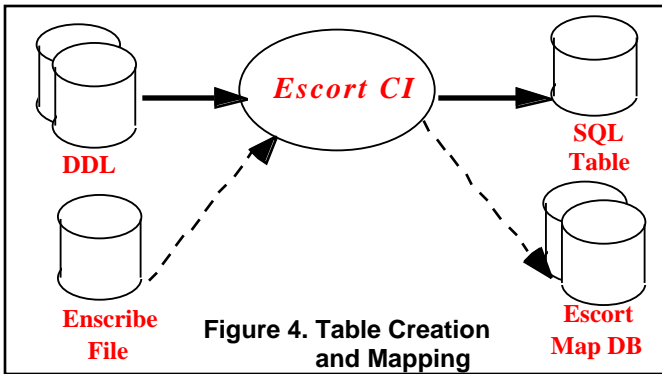


Figure 4. Table Creation and Mapping

Figure 4 illustrates processing the Enscribe DDL to create one or more tables and indexes from an Enscribe record definition. The user can specify a large number of options to customize and restructure the target database, including data type conversion and normalization. Physical attributes (extent sizes, partitioning, etc.) may be specified from the existing Enscribe database.

The mapping of the original Enscribe record to the SQL table(s) is stored in the mapping database; the DDL is no longer needed by the gateway. We have created a *mapped table*, which can be opened by a prepared Enscribe program.

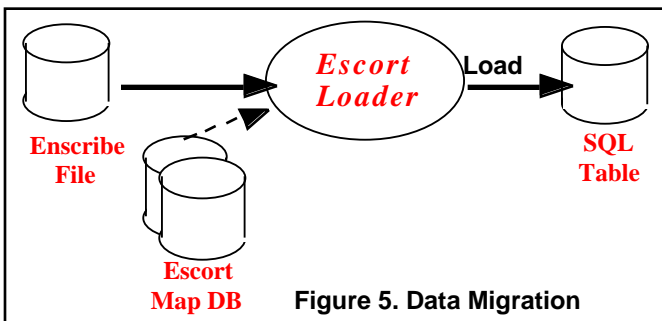


Figure 5. Data Migration

The Escort loader, shown in Figure 5., uses the information in the mapping database to migrate the data in the Enscribe file to the SQL tables. It automatically performs all the customized data type transformation and normalization specified in the table creation.

Finally, the prepared applications may be executed; the runtime passes through all operations for ordinary Enscribe files, but emulates operations for mapped SQL tables.

Escort Product Features

Multi-language support. Since Escort emulates the full Enscribe file system, it naturally supports all Tandem languages: COBOL, C, TAL, PASCAL. No source is required, but the programs must not be stripped. Enform and Enable can access the SQL database with no changes to the Tandem-supplied object code or user queries.

After preparation, the logic of the application is unchanged. Even when debugging at the lowest level, the same machine instructions are executed with the same results. Memory allocation is unchanged; Escort requires no primary, secondary, or extended global memory.

SQL Table Creation and Mapping. DDL descriptions of Enscribe data are processed to create SQL tables and to map Enscribe files to those tables. A simple conversion can be performed rapidly, but major restructuring will require design and experimentation.

The gateway converts Enscribe structured files: entry-sequenced, relative, and key-sequenced. Alternate key files are converted to SQL indexes.

Legacy Data Mapping. Most Enscribe databases have legacy data formats, such as YYMMDD dates. Built-in data types automatically convert legacy data fields to standard SQL data types. The built-in data types can be extended with custom, user-written, data types.

Non-Relational Record Mapping. Existing data records that cannot be described by DDL are transformed to a relational form with a custom record mapping. Thus, various forms of compressed data can be mapped to SQL.

Normalization. One-to-many table mappings convert a legacy Enscribe file to a normalized collection of SQL tables. This is commonly used to map the DDL REDEFINES or OCCURS clauses.

Data Migration. Although SQL provides an efficient bulk data loading facility, it does not support the data transformations and normalization supported by the gateway. Thus, Escort provides an enhanced facility to load the new SQL tables from the Enscribe files. When an Enscribe file has been normalized, many SQL tables are loaded in parallel from a single Enscribe file. For fallback and data distribution requirements, Enscribe files may be reverse loaded from the mapped SQL tables.

Application Execution. In prepared applications, the original object code is unchanged and the program executes in the same manner as before. Pathway application startup procedures and batch job execution require no significant change. Enform and Enable continue to operate as before, but the underlying database may be mapped SQL tables.

Hybrid Programming. Performance-critical programs can be modified to use advanced SQL. Native SQL and Enscribe I/O for the same table can coexist in a program. Hybrid programs need to be SQLCOMPIled as any other SQL program.

Compatibility

The emulation of Enscribe is nearly complete; all 58 relevant Enscribe procedures are intercepted. Only file checkpointing procedures are omitted.

All SQL errors are converted to appropriate Enscribe error codes; the Enscribe application should see only the relevant Enscribe error codes.

Data validation. Since SQL performs validation on each column value, operations that update the database with invalid data will be rejected; gateway operations that supply invalid data are rejected with an Enscribe error code.

The user may choose to deal with data validation errors in two ways: (1) allow the applications to fail and force the programmers to eliminate the source of invalid data, or (2) configure either built-in or custom data-scrubbing routines to convert invalid data to valid data during application execution. Escort analyzes invalid updates and indicates the particular fields in error.

Unstructured access and FAST-IO. Although unstructured I/O is not supported, Escort will correctly execute COBOL and CRE programs that request FAST-IO, but with normal efficiency.

Database Management. Neither FUP nor Enscribe database management can alter file partitioning and alternate key files on a mapped SQL table. The user must use SQLCI or other SQL-based tools to perform such operations. BACKUP and RESTORE have special options and requirements for SQL tables.

Performance

Excellent performance is a primary criteria for the gateway, as it is designed to be used for migrating high-performance Enscribe-based OLTP applications to SQL. Since Enscribe is one of the most efficient databases in existence, this is not an easy task. Considerable effort has been expended to optimize the emulator; emulation data structures and algorithms are all designed for the utmost speed. There remains, however, a need for the user to measure and manage the performance of the migrated application.

The performance effect can be separated into three categories:

Operations not requiring emulation. When operating on files that have not been migrated, the performance effect of the gateway is negligible. Each ordinary Enscribe operation is passed-through with no significant overhead.

Initial emulated operations. The gateway compiles dynamic SQL statements when the application program makes Enscribe requests on a SQL table, but these compilations are cached and reused for all subsequent operations that are similar. Typical statement compilations times are 0.1 to 0.3 seconds; typical programs compile one or two statements for each file open.

Mainline emulated operations. Gateway performance is equivalent to a *direct* translation of the program to hand-coded, compiled, SQL. Dynamic SQL statements are just as efficient as pre-compiled SQL statements. Once compiled, dynamic and embedded SQL statements are executed in the same fashion.

The requirement to manage performance is the direct result of two basic facts: (1) The gateway must emulate individual Enscribe operations with rather simple SQL statements; it uses sequential cursors to enhance buffering, but does not employ joins or complex set operations. (2) Simple SQL statements use more cpu than equivalent Enscribe operations; while Enscribe performs a simple move of the record, SQL performs column-by-column data transformation and validation.

In order to significantly improve on the gateway's performance, one cannot just replace simple Enscribe operations with simple SQL statements; some algorithm redesign is required to make use of efficient joins, set operations and blind updates.

As noted above, the gateway supports hybrid programs, so only the performance-critical parts of performance-critical programs need to be altered. Even when performance is an issue, the gateway requires a minor amount of reprogramming.

Escort Optimizations

When an application accesses SQL tables through the gateway, the gateway employs sophisticated techniques to reduce overhead:

Passthrough Mode. Due to a proprietary stack management scheme, the basic overhead for executing an Escort-prepared program is negligible. Applications that access only Enscribe files through the Escort gateway will see insignificant increased resource usage. Likewise, Escort has no effect on the execution of compiled SQL.

Map caching. When the gateway runtime requires map information stored in the Escort map database, the actual database fetch is eliminated by caching the map information in the monitor process. An OPEN is processed with a single interprocess message, and no actual database I/O.

Statement caching. Escort maintains a compiled SQL statement cache with a proprietary tagging algorithm to efficiently eliminate unnecessary recompilation.

Cursor management. Escort carefully manages cursors to obtain the best possible buffering performance from SQL. It also provides methods for the user to customize buffering, without changing the application program.

Data compression. Numeric PIC 9 fields and fixed-length ASCII text can be mapped to NUMERIC (i.e., binary) and VARCHAR, respectively. The user cannot detect the difference, but the resulting SQL table may be much smaller than the Enscribe file. This greatly reduces the amount of disk I/O and allows more data to be cached in memory.

Incremental Performance Management

A complete Enscribe-to-SQL migration, using either gateway or traditional technology, may result in a significant increase in cpu utilization, depending, of course, on how intensively the application accesses the database. If an application spends only 10 to 20% of its time performing database operations, the increase may not be significant.

With a gateway, one should take full advantage of incremental migration to manage risk, data migration outage, as well as performance. Migrate a single file and measure utilization and response time; make the necessary adjustments for that file before proceeding to the next. The effect of each small change can be monitored and addressed before overall system performance is affected.

Incremental migration also permits the user to make reasonable tradeoffs between hardware and software investment. The simplest solution to increased cpu utilization may be to add cpu resources in the form of a hardware upgrade, but another approach may be to reprogram some parts of the application. Where an Enscribe program may need to read and examine every record in order to update a small subset, a SQL set operation is performed efficiently at the disk process level. SQL also allows the "blind update" operation, making it unnecessary to read a record before updating it.

With Escort, this sort of performance optimization is made considerably easier because one can reprogram very small parts of a legacy program. Performance-critical routines can be hand-coded with compiled SQL, while the others can continue to access the data using Enscribe calls through the gateway.

The Escort Gateway and Year 2000

The past two issues of *Tandem Connection* presented comprehensive articles on the Year 2000 century rollover problem (Highlyman, Allen). The Escort gateway can make Year 2000 re-engineering

simpler and less risky, and provide a useful database technology upgrade at the same time.

First, built-in datatypes convert legacy dates in the Enscribe data to millennium-compliant dates in SQL. During the conversion process, the user simply designates certain fields as having, for example, a YYMMDD format. The corresponding SQL field will be of type DATE—including a century. As existing applications read or write the data, it will automatically be converted using a 100-year window algorithm.

The user may choose different 100-year windows for each field. For example, a birth date might have a 1910-2009 window, while a mortgage maturity date might have a 1980-2079 window.

Thus, the underlying database is made millennium compliant but the legacy programs execute as before; this does not re-engineer the programming logic. The gateway approach allows the user to define an *alternate* "Enscribe" mapping on each database table. The alternate mapping converts dates with centuries (YYYYMMDD). Programs can be converted, one-by-one, to use the new DDL and perform date calculations with centuries.

Although this approach does not eliminate all reprogramming, it has three important advantages: (1) It immediately upgrades the database to be millennium compliant and does not wait for all existing applications to deal with YYYYMMDD dates. (2) It allows programs that do not perform calculations on dates to be left as-is; this may be 80 to 90% of all programs. (3) The remaining programs can be converted incrementally; the most critical can be converted immediately.

Conclusion

Migrating a legacy database to SQL has many advantages, but high cost and risk have prevented the majority of Enscribe database users from making the transition. With the availability of an Enscribe-to-SQL database gateway, Tandem users have a new option that reduces those costs and risks to an acceptable level. This gateway can reduce the migration time from years to months and allows the advantages of SQL to be realized immediately.

Bibliography

- Alleman, R., Performance Challenges in Enscribe File Conversions, in *Tandem Connection* Vol. 18 No. 3, 1997
- Allen, L., Testing the Year 2000 Fix, in *Tandem Connection* Vol. 18 No. 3, 1997
- Brodie, M., and Stonebraker M. Migrating Legacy Systems: Gateways, Interfaces & the Incremental Approach, Morgan Kaufmann, San Francisco, 1995
- Highlyman, B., The Year 2000 Virus and Its 10-Step Cure, in *Tandem Connection* Vol. 18 No. 2, 1997

Comments or questions about this article may be addressed to the author at carr_richard@Tandem.com